

Please type a plus sign (+) inside this box → ☐

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No. **ST9-99-146** Total Pages **86**
First Named Inventor or Application Identifier
Jan Burchhardt
Express Mail Label No. **EL352820239US**

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

1. ☒ Fee Transmittal Form
(Submit an original, and a duplicate for fee processing)
2. ☒ Specification [Total Pages **61**]
(preferred arrangement set forth below)
 - Descriptive title of the invention
 - Cross References to Related Application
 - Statement Regarding Fed sponsored R & D
 - Reference to Microfiche Appendix
 - Background of the invention
 - Brief Summary of the invention
 - Brief Description of the Drawings (if filed)
 - Detailed Description
 - Claim(s)
 - Abstract of the Disclosure
3. ☒ Drawing(s) (35 USC 113) [Total Sheets **9**]
4. Oath or Declaration [Total Pages **4**]
 - a. ☒ Newly executed (original or copy)
 - b. ☐ Copy from a prior application (37 CFR 1.63(d))
(for continuation /divisional with Box 17 completed)
(Note Box 5 below)
 - i. ☐ **DELETION OF INVENTOR(S)**
Signed statement attached deleting inventor(s)
named in prior application, see 37 CFR
1.63(d)(2) and 1.33(b).
5. ☐ Incorporation by Reference (useable if Box 4b is checked)
The entire disclosure of the prior application, from which
a copy of the oath or declaration is supplied under Box
4b is considered as being part of the disclosure of the
accompanying application and is hereby incorporated by
reference therein.

ADDRESS TO: Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

6. ☐ Microfiche Computer Program (Appendix)
7. Nucleotide and/or Amino Acid Sequence Submission (if
applicable, all necessary)
 - a. ☐ Computer Readable Copy
 - b. ☐ Paper Copy (identical to computer copy)
 - c. ☐ Statement verifying identify of above copies

ACCOMPANYING APPLICATION PARTS

8. ☒ Assignment Papers (cover sheet & document(s))
9. ☐ 37 CFR 3.73(b) Statement ☐ Power of Attorney
(when there is an assignee)
10. ☐ English Translation Document (if applicable)
11. ☒ Information Disclosure ☒ Copies of IDS
Statement (IDS)/PTO-1449 Citations
12. ☐ Preliminary Amendment
13. ☒ Return Receipt Postcard (MPEP 503)
(Should be specifically itemized)
14. ☐ Small Entity ☐ Statement filed in prior application,
Statement(s) Status still proper and desired
15. ☐ Certified Copy of Priority Document(s)
if foreign priority is claimed
16. ☒ Other: Express Mail Certificate

17. If a CONTINUING APPLICATION, check appropriate box and supply the requisite information:

☐ Continuation ☐ Divisional ☒ Continuation-in-part (CIP) of prior application No.: **60/151,479**

18. CORRESPONDENCE ADDRESS

☒ Customer Number or Bar Code Label **21552** or ☐ Correspondence address below

NAME	Kory D. Christensen			
ADDRESS				
CITY	STATE	ZIP		
COUNTRY	TELEPHONE	FAX		

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, D.C. 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, D.C. 20231.

Express Mailing Label No. EL352820239US

PATENT APPLICATION
IBM No. ST9-99-146

UNITED STATES PATENT APPLICATION

of

JAN BURCHHARDT

and

SHYH-MEI HO

for

REPRESENTING IMS MESSAGES AS XML DOCUMENTS

MADSON & METCALF, P.C.

ATTORNEYS AT LAW
900 GATEWAY TOWER WEST
15 WEST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84101

REPRESENTING IMS MESSAGES AS XML DOCUMENTS

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to transaction processing systems. More particularly, the present invention relates to a system and method for representing IMS messages as XML documents.

Related Applications

This application claims the benefit of U.S. Patent Application No. 60/151,479, filed August 30, 1999, for "IMS Messages in XML," which is incorporated herein by reference.

Identification of Copyright

A portion of the disclosure of this patent document contains material subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Relevant Technology

With the explosive growth of the Internet, most of the world's computer systems are now interconnected or capable of being interconnected. However, in order to share data, the systems need to understand each other's data formats. In recent years, the computer industry has evolved at such a rapid pace that systems developed only a few years apart use vastly different and incompatible formats. Such incompatibility problems tend to increase with the "age" differences of the systems.

The Information Management System (IMS) is one of the oldest and perhaps the most popular transaction processing (TP) systems. A TP system supervises the sharing of resources for concurrently processing multiple transactions, such as queries to a database. Anyone who has ever used an ATM, rented a car, or booked a flight, has probably used IMS.

IMS was developed by IBM in the 1960's as a inventory tracking system for the U.S. moon landing effort. Today, interfacing IMS with newer systems, particularly with systems of different manufactures over the Internet, is problematic.

As illustrated in Figure 1, an IMS typically includes two major components: an IMS Transaction Monitor (IMS/TM), which is responsible for scheduling, authorization, presentation services and operational functions, and a hierarchical database, DL/1. Both components are independently configurable.

1 For example the IMS/TM 12 can use a relational database, such as DB/2, rather than
2 DL/1. The various components of an IMS 10 communicate via the MVS operating
3 system 16.

4 As illustrated Figure 2, the architecture of IMS is divided into four regions:
5 a message processing region (MPR) 20, a batch message processing (BMP) 22 region,
6 an interactive fast path (IFP) 26 region, and an IMS control (IMSCTL) 24 region.
7 The MPR 20 is used to execute message-driven applications 18. Execution of
8 applications 18 in this region 20 is triggered by incoming messages, such as those
9 received from a terminal.

10 By contrast, applications 18 in the BMP 22 are not message driven. They are
11 usually scheduled to run at times of low system activity, such as nights and
12 weekends. Typically, such applications 18 perform a number of predefined
13 operations, after which they immediately terminate.

14 The IFP 24 allows fast and simple requests to the hierarchical database 14.
15 Applications 18 operating in the IFP 24 bypass the normal scheduling mechanism,
16 providing relatively fast response times. In general, IFP applications 18 stay
17 resident even if they are not needed.

18 The IMSCTL 26 is responsible for overseeing all TP tasks, as well as for
19 controlling all dependent regions (e.g., MPR 20, BMP 22, and IFP 24). Essentially,
20 the IMSCTL 26 has three main responsibilities: telecommunications, message
21 scheduling, and logging/restart.

1 For example, as illustrated in Figure 3, the IMSCTL 26 controls one or more
2 connected terminals 28, sending and receiving messages to and from the terminals
3 28. Moreover, the IMSCTL 26 logs all transactions in order to provide the capability
4 of undoing non-committed transactions in the event of a system failure.

5 In addition, every time the IMSCTL 26 receives an input message 30 from a
6 terminal 28, it schedules an application 18 to process the message 30. The IMSCTL
7 26 identifies the desired application 18 and puts the message 30 in the application's
8 message queue 32. The application 18 processes the message 30 and responds to the
9 originating terminal 28 by placing an output message 30 in the terminal's message
10 queue 34.

11 As illustrated in Figure 4, an input message 30 typically includes the
12 following fields:

13	LL	Length of the message segment.
14	ZZ	Reserved for IMS.
15	TRANCODE	Transaction code that identifies the application 18.
16	Text	Message text sent from the terminal 28 to the
17		application 18.

18 The structure of an output message 30 is similar, except that the TRANCODE field
19 is missing.

20 In general, messages 30 belong to one particular IMS application 18. When
21 the application 18 is implemented, the format of the message 30, including the types

1 and lengths of its fields, must be defined. The format of a message 30 is referred to
2 herein as a message definition 38. Message definitions 38 may be implemented
3 using various programming languages, such as COBOL, assembler, PL/I and
4 Pascal. For example, the message definition 38 illustrated in Figure 4 is
5 implemented in COBOL.

6 Unfortunately, IMS messages 30 are in a proprietary format, whereas the
7 Internet is based on open standards, such as the HyperText Markup Language
8 (HTML), a variant of the eXtensible Markup Language (XML). As a result,
9 interfacing IMS with remote systems via the Internet can be difficult. Accordingly,
10 what is needed is a system and method for representing IMS messages 30 in an
11 open, interchangeable format, such as XML.

SUMMARY OF THE INVENTION

The present invention solves many or all of the foregoing problems by providing a system and method for representing IMS messages as XML documents.

In one aspect of the invention, a method includes generating an XML document template from an IMS message definition and merging an IMS message with the generated template to produce a corresponding XML document.

In another aspect, a process of generating the XML document template includes obtaining an IMS message definition; obtaining a DTD for representing arbitrary IMS message definitions; compiling the IMS message definition with an option configured to produce an associated data (Adata) file; and parsing the Adata file using the DTD to generate an XML document template corresponding to the IMS message definition.

In various embodiments, the IMS message definition may include program source code in a language selected from the group consisting of COBOL, PL/I, Assembler, and Pascal. Additionally, the Adata file may include at least one IMS message definition in a relatively language independent format compared with the program source code.

In another aspect, a process of obtaining the DTD may include creating a UML object model for representing arbitrary IMS message definitions; and processing the object model using an XMI utility to generate the DTD.

1 In still another aspect, a process of merging the XML document template
2 with the IMS message may include identifying a placeholder within XML document
3 template for receiving a corresponding value from the IMS message; reading the
4 value from the IMS message; and inserting the value into a location within the XML
5 document template indicated by the placeholder. In certain embodiments, the
6 placeholder may be implemented as an XML tag.

7 In still another embodiment of the invention, a placeholder may include an
8 associated tag for indicating that a corresponding value exists within the IMS
9 message. Additionally, a placeholder may include an associated tag for indicating
10 the size of the corresponding value within the IMS message.

11 In yet another aspect, a system for representing IMS messages as XML
12 documents may include a template generation module configured to generate an
13 XML document template from an IMS message definition; and a merging module
14 configured to merge an IMS message with the generated template to produce a
15 corresponding XML document.

16 In various embodiments, the template generating module may include a
17 compiler configured to compile an IMS message definition with an option
18 configured to produce an associated data (Adata) file; and a parser configured to
19 parse the Adata file using a DTD for representing arbitrary IMS message definitions
20 to generate an XML document template corresponding to the IMS message
21 definition.

1 In certain embodiments, the system may also include a modeling tool
2 configured to create a UML object model for representing arbitrary IMS message
3 definitions; and an XMI utility for generating the DTD from the UML object model.

4 These and other objects, features, and advantages of the present invention
5 will become more fully apparent from the following description and appended
6 claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is more fully disclosed in the following specification, with reference to the accompanying drawings, in which:

Figure 1 is a schematic block diagram of an Information Management System (IMS);

Figure 2 is a schematic block diagram of IMS processing regions;

Figure 3 is a schematic block diagram of message processing within an IMS;

Figure 4 is a schematic block diagram of the structure of IMS messages;

Figure 5 is a schematic block diagram of a technique for representing an IMS message definition in XML according to an embodiment of the invention;

Figure 6 is a schematic block diagram of an alternative technique for representing an IMS message definition in XML according to an embodiment of the invention;

Figure 7 is a schematic block diagram of a system for representing IMS messages as XML documents according to an embodiment of the invention;

Figure 8 is a schematic block diagram of a UML object model of an IMS message definition according to an embodiment of the invention;

Figure 9 is a class hierarchy of a parser according to an embodiment of the invention;

1 Figure 10 is a schematic block diagram of a technique for representing an IMS
2 message definition as an XML document template according to an embodiment of
3 the invention;

4 Figure 11 is a schematic flowchart of a method for representing IMS messages
5 as XML documents according to an embodiment of the invention;

6 Figure 12 is a schematic flowchart of a process for generating an XML
7 document template from an IMS message definition according to an embodiment
8 of the invention; and

9 Figure 13 is a schematic flowchart of a process for merging an XML
10 document template with an IMS message according to an embodiment of the
11 invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Certain presently preferred embodiments of the invention are now described with reference to the Figures, where like reference numbers indicate identical or functionally similar elements. The components of the present invention, as generally described and illustrated in the Figures, may be implemented in a variety of configurations. Thus, the following more detailed description of the embodiments of the system and method of the present invention, as represented in the Figures, is not intended to limit the scope of the invention, as claimed, but is merely representative of presently preferred embodiments of the invention.

Throughout the following description, various system components are referred to as "modules" or the like. In certain embodiments, these components may be implemented as software, hardware, firmware, or any combination thereof.

For example, as used herein, a module may include any type of computer instruction or computer executable code located within a memory device and/or transmitted as electronic signals over a system bus or network. An identified module may include, for instance, one or more physical or logical blocks of computer instructions, which may be embodied within one or more objects, procedures, functions, or the like.

The identified modules need not be located physically together, but may include disparate instructions stored at different memory locations, which together implement the described logical functionality of the module. Indeed, a module may

1 include a single instruction, or many instructions, and may even be distributed
2 among several discrete code segments, within different programs, and across
3 several memory devices.

4 5 Obtaining Message Definitions from an Application

6 As noted, IMS messages 30 are defined within a corresponding application
7 18. As such, in order to generate an XML document that represents a message 30,
8 a technique for obtaining a message definition 38 from an application 18 would be
9 highly desirable.

10 In certain embodiments, the format of a message 30 may be directly obtained
11 from an application's source code using a specially-designed parser. For example,
12 the following is a typical COBOL message definition 38, which may be analyzed by
13 the parser:

```
14      01  INPUT-MSG.  
15          02  IN-LL          PICTURE IS 9(2) .  
16          02  IN-ZZ          PICTURE IS 9(4) .  
17          02  IN-TRAN        PICTURE IS X(10) .  
18          02  IN-COMMAND     PICTURE IS X(8) .  
19          02  TEMP-COMMAND   REDEFINES IN-COMMAND .  
20              04  TEMP-IOCMD          PIC X(3) .  
21              04  TEMP-FILLER         PIC X(5) .
```

22 However, COBOL has a very complex syntax. For example, variable
23 definitions in COBOL may include complex dependencies, making the source code
24 of message definitions 30 difficult to parse. Moreover, because each language is

1 different, a different parser would typically be required for each different
2 programming language, such as PL/I and Assembler.

3 Accordingly, in one embodiment of the invention, message definitions 38 are
4 obtained from a System Associated Data (SysAdata) file 78 (illustrated in Figure 7),
5 which is produced by various compilers 76 when the "adata" option is specified.
6 In essence, the SysAdata file 78 is a compiler-generated debugging aid. It contains
7 information about the structure of a program, the contained data, the data flow
8 within the program, and the system in which the program operates.

9 One reason for relying the SysAdata file 78 rather than the application source
10 code 74 is that a single parser may be used for different programming languages.
11 For example, PL/I compilers and some high-level assemblers also generate
12 SysAdata files 78. Although some differences exist in the SysAdata files 78
13 produced by different compilers 76, such differences may be compensated for by
14 those skilled in the art.

15 A record of the SysAdata file 78 generally includes the following two
16 sections:

- 17 1. a 12 byte header section, which has the same structure for all record
18 types; and
- 19 2. a variable length data section, which varies by record type.

20 To extract a message definition 38 from a SysAdata file 78, not all record types are
21 needed. For example, in one embodiment, only the Common Header Section, the

Compilation Unit Start/End Record, and the Symbol Record are used. These records are described in greater detail below.

The format of the various records within the SysAdata file 78 are shown in Tables 1-3, with the following abbreviations being used to indicate data types:

- c** indicates character (EBCDIC or ASCII) data;
- h** indicates 2-byte binary integer data;
- f** indicates 4-byte binary integer data;
- x** indicates hexadecimal data or 1-byte binary integer data with the length of the number given behind the data type.

Common Header Section

The Common Header Section contains, among other things, a record code that identifies type and length of the record. The 12 byte header section has the following format:

Table 1

Field	Size	Description
Language Code	x 1	16 High Level Assembler
		17 COBOL on all platforms
		40 PL/I on supported platforms
Record type	h 2	x 0000 Job Identification record
		x 0001 ADATA Identification record
		x 0002 Compilation unit start/end
		record
		x 0010 Options record

1		x 0020	External Symbol record
		x 0024	Parse Tree record
2		x 0030	Token record
		x 0032	Source Error record
3		x 0038	Source record
		x 0039	COPY REPLACING record
4		x 0042	Symbol record
		x 0044	Symbol Cross-Reference record
5		x 0046	Nested Program record
		x 0060	Library record
6			
		x 0090	Statistics record
7		x 0120	EVENTS record
8	Adata architecture level	x 1 3	Definition level for the header structure
9	Flag	x 11.	Adata record integer are in Little Endian (Intel) format
10	1	This record is continued in the next record
11		1111 11..	Reserved for future use
12	Adata record edition level	x 1	Used to indicate a new format for a specific record type. Usually zero.
13	Reserved	c 4	Reserved for future use
	Adata field length	h 2	The length in bytes of the data following the header.

Compilation Unit Start/End Record

The Compilation Unit Start/End Record is the second and the last record in the SysAdata file 78. The 8 byte record uses the following format:

Table 2

Field	Size	Description
Type	h 2	Compilation unit type, which can be one of the following:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

x0000

Start compilation unit

x0001

End compilation unit

Reserved	c 2	Reserved for future use
----------	-----	-------------------------

Reserved	f 4	Reserved for future use
----------	-----	-------------------------

Symbol Record

The Symbol Record contains all of the information needed to understand the structure of a message definition 38 that has been compiled into a SysAdata file 78. Only the fields that are used in a presently preferred embodiment of the invention are listed in Table 3. For simplicity, Table 3 only indicate the size, and not the position, of the fields. The position of the fields can be determined from the source code of Appendix A by those skilled in the art.

Table 3

Field	Size	Description														
Symbol ID	f 4	Unique ID of symbol														
Level	f 4	True leve number of symbol (or relative level number of a data item within a structure). Level is in the range of 01-49, 66 (for rename items), 77, or 88 (for condition items).														
Symbol Type	x 1	<table><tr><td>x 68</td><td>Class name</td></tr><tr><td>x 58</td><td>Method name</td></tr><tr><td>x 40</td><td>Data name</td></tr><tr><td>x 20</td><td>Procedure name</td></tr><tr><td>x 10</td><td>Mnemonic name</td></tr><tr><td>x 08</td><td>Program name</td></tr><tr><td>x 81</td><td>Reserved</td></tr></table>	x 68	Class name	x 58	Method name	x 40	Data name	x 20	Procedure name	x 10	Mnemonic name	x 08	Program name	x 81	Reserved
x 68	Class name															
x 58	Method name															
x 40	Data name															
x 20	Procedure name															
x 10	Mnemonic name															
x 08	Program name															
x 81	Reserved															
Symbol Attribute	x 1	Numeric														

	x 2	Alphanumeric
	x 3	Group
		note: other attributes are ignored
Clauses	x 1	For numeric, alphanumeric and group items:
	1... ..	Value
	.1.. ..	Indexed
	..1.	Redefines
	...1	Renames
	... 1...	Occurs
1..	Has Occurs Keys
1.	Occurs Depending on
1	Occurs in Parent
		note: other values are ignored
Data Flags1	x 1	1... .. Redefined
	.1.. ..	Renamed
	..1.	Synchronized
	...1	Implicitly redefined
	... 1...	Date field
1..	Implicit redefines
1.	FILLER
1	Level 77
Size	f 4	the size of this data item. The actual number of
		bytes this item occupies in storage.
		Also referred to as the "Length Attribute."
Parent ID	f 4	The symbol ID of the immediate parent of the
		symbol being defined.
Redefined ID	f4	The symbol ID of the data item, that this item
		renames.
Symbol Name Length	h 2	The number of characters in the symbol name.

The source code of the application 18 and the SysAdata file 78 contain essentially the same information about the message definition 38. However, in

1 order to read the source code, a parser would need to understand a subset of the
2 COBOL language.

3 By contrast, the SysAdata file 78 has a clearly defined format, each bit having
4 a definite meaning. Consequently, the SysAdata file 78 may be easier to read and
5 understand, and the corresponding parser more simple and easy to implement.

6 As shown in Figure 7, a compiler 76 produces the SysAdata file 78 when it
7 is able to compile an application source code file 74 without major errors.
8 Accordingly, a system and method in one embodiment of the invention verifies that
9 the compilation completed with a return code of 4 or less. Analysis can still proceed
10 if "Information" and "Warning" messages are generated, but there should be no
11 "Error", "Severe error", or "Termination" messages.

12 In certain embodiments, compiling a subset of the application 18, i.e. the
13 message definition 38, itself, is advantageous. For example, a particular message
14 definition 38 may be extracted from the application's source code 74 or a copy file
15 and copied into the working-storage section of a COBOL source file template for
16 separate compilation. In certain embodiments, a message definition extractor 75
17 may be provided for this purpose, which may extract a user-specified message
18 definition 38 from the source code 74 of an application 18. The extractor 75 may
19 create a valid source file for a compiler 76 as illustrated below:

20 IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE-MSG.
21 ENVIRONMENT DIVISION.
DATA DIVISION.

1 WORKING-STORAGE SECTION.

01 INPUT-MSG.

2 02 IN-LL PICTURE IS 9(2).

02 IN-ZZ PICTURE IS 9(4).

3 02 IN-TRAN PICTURE IS X(10).

02 IN-COMMAND PICTURE IS X(8).

4 02 TEMP-COMMAND REDEFINES IN-COMMAND.

04 TEMP-IOCMD PIC X(3).

5 04 TEMP-FILLER PIC X(5).

6 PROCEDURE DIVISION.

STOP RUN.

7 Of course, it would also be possible to read a SysAdata file 78 from the
8 compilation process of an entire application 18. However, since one application 18
9 may include a plurality of messages definitions 38 in the working-storage section,
10 a user would need to make a choice as to which message definition 38 to use.

11 Generating a Document Type Definiton (DTD) for IMS Messages

12 In order to represent IMS messages 30 in XML, generating a Document Type
13 Definition (DTD) 54 is highly desirable. Various techniques may be used to create
14 the DTD 54. For example, in certain embodiments, a different DTD 54 may be
15 created for each particular message 30. Such a DTD 54 would allow any XML
16 parser to understand the structure of the associated message 30. In alternative
17 embodiments, a generic DTD 54 for arbitrary messages 30 may be created. These
18 two options are fundamentally different and are described more fully in the
19 following sections.
20
21

One DTD Per Message Definition

As illustrated in Figure 5, each data entry 53 (e.g., variable or group) in the message definition 38 may be directly represented by an element in the DTD 54. Each element may have one or more attributes, which contain information about the data entry.

The benefit of this technique is that the corresponding XML documents 44 are relatively simple. For example, an XML document 44 corresponding to the message definition 38 of Figure 5 may include the following:

```
<IN-LL length="2" type="Numeric">55</IN-LL>  
<IN-ZZ length="4" type="Numeric">102</IN-ZZ>  
<IN-LASTNAME length="10" type="Alphanumeric">Meyer</IN-LASTNAME>
```

Although this approach results in a straightforward DTD 54 and simple XML documents 44, it has two major disadvantages. First, no tool can be effectively used to create the DTD 54. Rules governing DTD 54 creation typically need to be implemented in a parser. Second, because different names may be used for elements and attributes, no generalized techniques for reading and writing corresponding XML documents 44 may be provided.

Generic DTD for all Messages

In an alternative embodiment, as illustrated in Figure 6, a single, generic DTD 54 may be used for all IMS messages 30. In this case, each data entry 53 from a message definition 38 may be represented as a "DataItem" 55. The structure of a

1 DataItem 55 is defined in the DTD 54 of Appendix C, and is described in greater
2 detail below.

3 The generic DTD 54 approach offers several advantages. First, the generic
4 DTD 54 may be created using standard tools from a UML object model, as explained
5 below. Second, an XMI utility, such as IBM's XMI Toolkit™, may be used to provide
6 generic access methods for reading and writing XML documents 44. Third, the
7 generic DTD 54 is relatively language independent (compared to source code 74),
8 such that message definitions 38 implemented in Assembler, PL/I, or the like can
9 be presented in the same way. Finally, the DTD 54 may be maintained and updated
10 in a common location.

11 Using the generic DTD 54 approach, as illustrated in Figure 6, a message
12 definition 38 may be transformed into an XML document template 58 that
13 represents the format of a particular message 30. Later, a merging module 80 may
14 be used to merge the actual IMS message 30 with the template 58 to create an XML
15 document 44 that represents the message 30. This process is described in greater
16 detail below.

17 Modeling IMS Message Definitions

18 Figure 7 illustrates a system 60 for generating the above-described DTD 54
19 and XML document templates 58 according to an embodiment of the invention. The
20 system 60 may include a Uniform Modeling Language (UML) modeling tool 62.
21

1 The UML modeling tool 62 may be used to produce a generic UML object model 64
2 to represent arbitrary IMS messages 30. In one embodiment, the UML modeling
3 tool 62 is Rational Rose™, a visual modeling tool available from Rational Software.

4 UML is a language for specifying, visualizing, constructing, and
5 documenting software systems, in addition to business models and the like. UML
6 is capable of representing the static, dynamic, environmental, and organizational
7 aspects of almost any system.

8 In order to generate a DTD 54, the abstract structure of the message 30 should
9 be modeled. As noted above, in one embodiment, each data entry 53 of the
10 message definition 38 may be represented as a *DataItem* 55, whether the entry is a
11 group, a redefined variable, or a normal variable. For example, if a data entry 53
12 is a group, all group members are also *DataItems* 55, but they are one level lower
13 in the object hierarchy of the model.

14 Figure 8 illustrates a UML object model 64 of an IMS message definition 38
15 according to one embodiment of the invention, as implemented by the UML
16 modeling tool 62. In various embodiments, the model 64 includes a *DataItem* class
17 65 for representing each data entry 53 of the message definition 38. An instance of
18 the *DataItem* class 65 stores data retrieved from a Symbol Record of a SysAdata file
19 78, as explained below.

20 In various embodiments, the *DataItem* class 65 includes a number of
21 attributes:

1 Name: Contains the name of the data entry 53 represented by an instance of
2 the class.

3 Type: Contains the Symbol Attribute. Valid values are defined in Class
4 tSymbolAttribute (see Table 3).

5 Length: Contains the length of the DataItem instance. If the data entry 53 is
6 a group or the root element, the length is added to the length of all of
7 the children.

8 hasValue: Indicates if the DataItem instance is used to store actual data or used
9 to group other DataItems together. It also indicates whether the
10 <Value> tag is present.

11 Value: Contains the value of the variable.

12 As illustrated in the object model 64 of Figure 10, a DataItem instance may have zero
13 or more children. The children are also DataItem instances and are contained within the
14 parent. Every DataItem instance has zero or one parent. Accordingly, the parent-child
15 hierarchy of the object model 64 may be used to model the hierarchical structure of IMS
16 message definitions 38.

17 In various embodiments, the classes tSymbolAttribute, tString and tBoolean may
18 serve as type classes for the DataItem attributes. As such, attributes of the class
19 tSymbolAttribute become possible values of the DataItem.Type attribute.

20 Referring again to Figure 7, the system 60 may also include an XML Metadata
21 Interchange (XMI) utility 66, such as XMI Toolkit™, available from IBM. XMI is an

open standard released by the Object Management Group (OMG) for simplifying the exchange of data and metadata between different products from different vendors. IBM's XMI Toolkit is written entirely in Java and offers interfaces to facilitate incorporation into other projects or products. However, languages other than Java may be used in various implementations.

In one embodiment, the XMI utility 66 automatically generates the DTD 54 from the UML object model 64. The following is a portion of a DTD 54 for IMS messages 30 according to an embodiment of the invention. A more complete DTD 54 including XMI-specific additions may be found in Appendix C.

```
<!ELEMENT DataItem (DataItem.Name?, DataItem.Type?, DataItem.Length?,
                    DataItem.hasValue?, DataItem.Value?, XMI.extension*,
                    DataItem.parent?, DataItem.child*)? >
<!ATTLIST DataItem
            %XMI.element.att;
            %XMI.link.att; >

<!ELEMENT DataItem.parent (DataItem)? >

<!ELEMENT DataItem.child (DataItem)* >

<!ELEMENT DataItem.Name (#PCDATA | XMI.reference)* >

<!ELEMENT DataItem.Type EMPTY >
<!ATTLIST DataItem.Type xmi.value ( Root | Numeric | Alphanumeric | Group
) #REQUIRED >

<!ELEMENT DataItem.Length (#PCDATA | XMI.reference)* >

<!ELEMENT DataItem.hasValue EMPTY >
<!ATTLIST DataItem.hasValue xmi.value ( true | false ) #REQUIRED >

<!ELEMENT DataItem.Value (#PCDATA | XMI.reference)* >
```

1 In addition, the XMI utility 66 may create a plurality of Java XMI document
2 access classes 68, which are used to read and write XML files based on the DTD 54,
3 as described more fully below.

4 5 Generating XML Document Templates

6 Referring again to Figure 7, the system 60 may also include a SysAdata parser 72,
7 which uses the generated DTD 54 and XMI classes 68 to parse the SysAdata file 78. As
8 previously noted, the SysAdata file 78 may be generated by a compiler 76, such as IBM's
9 Visual Age™ COBOL compiler, while compiling an application 18 from source code 74.
10 In one embodiment, the output of the parser 72 is an XML document template 58 that
11 represents the format of a particular IMS message 30. As used herein, the parser 72 and the
12 compiler 76 may be referred to collectively as a template generation module 77.

13 In alternative embodiments, however, the template generation module 77 may not
14 include the compiler 76. For example, the template generation module 77 may directly parse
15 and transform the message definition 38 into an XML document template 58.

16 As noted, the XMI utility 66 may also produce Java XMI classes 68 to read and write
17 XML files. Accordingly, the parser 72 may be implemented in Java, although the invention
18 is not limited in this respect. Figure 9 illustrates a class hierarchy 83 of the parser 72
19 according to an embodiment of the invention. The source code and a brief description of the
20 classes may be found in Appendices A and B, respectively.

21 Figure 10 further illustrates the above-described process of generating an XML

1 document template 58 from an IMS message definition 38. As illustrated, the
2 <DataItem.Value> tags are empty since no values from an IMS message 30 have been
3 supplied. Later, as described below, the IMS message 30 will be merged with the template
4 58 to create the XML document 44. The <DataItem.Value> tags function essentially as
5 “place holders” for receiving corresponding values from the IMS message 30.

6 Referring now to Figure 11, a schematic flowchart illustrates a method 90 for
7 representing IMS messages 30 as XML documents 44. In one embodiment, the
8 method 90 begins by generating 92 an XML document template 58 for the message
9 30 to be represented. Thereafter, the method 90 continues by merging an IMS
10 message 30 with the XML document template 58 to produce the XML document 44.

11 Figure 12 provides further details of the process 92 of generating the XML
12 document template 58. In one embodiment, the process 92 begins by extracting 96
13 a message definition 38 from the source code 74 of an IMS application 18 or an
14 associated copy file. Thereafter, the process 92 continues by compiling 98 the
15 extracted message definition 38 using the “adata” option. Finally, the process 92
16 concludes by parsing 100 the resulting SysAdata file 78 with the DTD 54 to produce
17 the XML document template 58.

18 Figure 13 shows the process 94 of merging the IMS message 30 with the XML
19 document template 58 in additional detail. The process 94 may begin by reading
20 102 the next DataItem 55 from the XML document template 58. Thereafter, a
21 determination 104 is made whether the <DataItem.hasValue> tag of the DataItem

1 55 has a "true" value. If not, the method returns to step 102 to read the next
2 DataItem 55.

3 Otherwise, the process 94 may continue by reading 106 a value (e.g. a
4 quantity of data) from the IMS message 30 equal in size to the value of the
5 <DataItem.Length> tag. For example, if the <DataItem.Length> tag has a value of
6 5, the process 94 may read 106 the next 5 bytes from the IMS message 30 in one
7 embodiment. Thereafter, the process 94 continues by inserting 108 the value read
8 from the message 30 into the XML document template 58, i.e. within the
9 <DataItem.Value> tags of the corresponding DataItem 55.

10 A determination 110 is then made as to whether more DataItems 55 remain
11 in the XML document template 58. If so, the process 94 returns to step 102 to read
12 the next DataItem 55. Otherwise, the process 94 is complete.

13 Based on the foregoing, the present invention offers a number of advantages
14 over conventional IMS interface systems. For example, the present invention allows
15 proprietary IMS messages 30 to be represented using openly interchangeable
16 documents, such as XML documents 44. The documents 44 may comply with the
17 latest XMI standard, which enables an IMS 10 to exchange data with a variety of
18 XMI-enabled devices, such as Web browsers and the like.

19 Importantly, XML documents 44 may be easily converted for display on a
20 variety of computing platform using the emerging XML Style Language (XSL)

21

1 standard. As such, the XMI to IMS interface is capable of replacing all other
2 interfaces between IMS and products from other vendors.

3 Additionally, because the SysAdata file 78 is used in one embodiment to
4 obtain IMS message definitions 38, the invention is not limited to a single
5 programming language as in conventional approaches. For example, a single
6 parser 72 may be used with COBOL, PL/I, and other compilers.

7 The present invention may be embodied in other specific forms without
8 departing from its scope or essential characteristics. The described embodiments
9 are to be considered in all respects only as illustrative and not restrictive. The scope
10 of the invention is, therefore, indicated by the appended claims rather than by the
11 foregoing description. All changes which come within the meaning and range of
12 equivalency of the claims are to be embraced within their scope.

13 What is claimed is:
14
15
16
17
18
19
20
21

Appendix ASource Code

```
1  /**
2
3  * Superclass for all classes
4  */
5  import java.io.*;
6  class Super {
7      static FileOutputStream fos;
8  public static void trace(String s) {
9      System.out.println(s);
10     }
11 }
12
13 /**
14 * Superclass for all record types. Provides methods to read adata in buffer.
15 */
16
17 import java.io.*;
18 import java.lang.reflect.*;
19 import java.lang.Class.*;
20
21 class Record extends Super{
22     int[] buffer;
23     int length;
24
25     /**
26     * reads data from the sysadata file and fills the objects buffer
27     */
28     public void fillBuffer(FileInputStream fis, int _length) {
29         buffer = new int[_length];
30
31         for (int i= 0; i<_length;i++){
32             try{
33                 buffer[i] = fis.read();
34             } catch(IOException e){
35                 trace("Error in "+this.getClass()+"."+e);
36             }
37         }
38     }
39 }
40
41 /**
42 * SysAdata reads and interprets a COBOL Associated Data (ADATA) file
43 * and creates the XMI document.
44 */
45
46 import java.io.*;
47 import java.util.Vector;
48 import com.ibm.ims.message.*;
49
50 public class SysAdata extends Super{
51     static FileInputStream fis = null;
```

```

1  static CHSRecord chs_rec;
   static CURecord cu_rec;

2  static Vector symbolVector = new Vector();
   static Id model = null;
3  static int rootLength = 0;
   /**
4  * addID adds one or more DataItem object to the model recursevily
   */
   public static int addID(Id parent, int symbolNumber){
5       SymbolRecord symbol = (SymbolRecord) symbolVector.elementAt(symbolNumber);
       try{
6           trace("adding "+symbol.getName());
           Id id = parent.add(Type.DATA_ITEM, null);
           id.set(Property.NAME, symbol.getName());
7           id.set(Property.TYPE, symbol.getAttribute());
           id.set(Property.LENGTH, symbol.getLength());
           id.set(Property.HAS_VALUE, ""+symbol.hasValue());
8           if(symbol.hasValue()){
               rootLength = rootLength + symbol.getSize();
               id.set(Property.VALUE, "");
9           }

           symbolNumber++;
10          if((symbol.hasValue() == false) && (symbolNumber < symbolVector.size())){
               SymbolRecord nextSymbol = (SymbolRecord)
11          symbolVector.elementAt(symbolNumber);
               if(nextSymbol.redefines()) {
12                   symbolNumber = addID(id,symbolNumber);
               }
               else {
13                   while ((symbol.getLevel() < nextSymbol.getLevel()) && (symbolNumber
               < symbolVector.size())){
14                       symbolNumber = addID(id,symbolNumber);
                       if(symbolNumber < symbolVector.size())
15                           nextSymbol = (SymbolRecord)
                           symbolVector.elementAt(symbolNumber);
16                   }
               }
               } catch(Exception e) {e.printStackTrace();}

17          // returns the number of the next symbol to be processed
           return symbolNumber;
18      }

   public static void main(String args[]){
19       boolean loopflag = true;
       int counter = 0;

20       if(args.length > 1){
           //open adata file which is given as first argument in the parameter list
           try {
21               fis = new FileInputStream(args[0]);
               //fis = new FileInputStream("d:/jan/parser/sysadata/data.adt");

```



```

1      }
2      catch(java.io.FileNotFoundException e){
3          System.out.println("error" + e);
4          return;
5      }
6
7      //reads first common header section (CHSRecord) from file
8      chs_rec = new CHSRecord(fis);
9
10     /**
11     * loop reads records from the adata file
12     * checks for CURecord, SymbolRecord and skips all other records
13     * stores SymbolRecords in symbolVector
14     */
15     do{
16         switch (chs_rec.getrecordType()){
17             case 2: {
18                 //Compilation Unit Record, 02hex
19                 cu_rec = new CURecord(fis);
20                 if(cu_rec.getType() == 1)
21                     loopflag = false;
22                 chs_rec.fill(fis);
23                 break;}
24             case 66: {
25                 //Symbol Record, 042hex
26                 S y m b o l R e c o r d   s y m b o l r e c o r d   =   n e w
27                 SymbolRecord(fis,chs_rec.getrecordLength());
28                 if(symbolrecord.getType()==64)
29                     symbolVector.addElement(symbolrecord);
30                 chs_rec.fill(fis);
31                 break;}
32             default: {
33                 //all other records are skipped
34                 skip(chs_rec.getrecordLength());
35                 chs_rec.fill(fis);
36                 break;}
37         }
38     }while(loopflag);
39
40     try{
41         fis.close();
42     }catch(IOException e){trace(""+e);}
43
44     try {
45         model = Model.instance().getSession();
46         Id root = model.add(Type.DATA_ITEM, null);
47         root.set(Property.NAME, "ROOT");
48         root.set(Property.TYPE, "ROOT");
49         root.set(Property.HAS_VALUE, "False");
50
51         do{
52             counter = addID(root, counter);
53         }while(counter<symbolVector.size());

```

```

1          root.set(Property.LENGTH, ""+rootLength);
          Model.instance().save(root, args[1], Model.DEFAULT, new java.util.Vector());
          //Model.instance().save(root, "sample.xml", Model.DEFAULT, new java.util.Vector());
2      } catch (Exception e) {e.printStackTrace();}
      }
3  else
          trace("No input and output file given\nSyntax is: java SysAdata input.adt output.xml");
      }
4  /**
   * skips the given amount of bytes in the SysAdata input file.
   */
5  public static void skip(int length) {
      try{
6          fis.skip(length);
          } catch(java.io.IOException e){trace(""+e);}
      }
7  }

8  /**
   * CHSRecord: Common header Section - x0001
   * 12byte long, common for all record types
9  */

10 import java.io.*;
   class CHSRecord extends Record{

11     public byte langCode; // Language Code
   public short recordType; // Record Type
12     public byte sysadateLevel;// SysAdata Architecture Level
   public byte contFlag; // bit 1: record is continued; bit 2: integers are Little-Endian; bits 3-8: reserved
13     public byte editionLevel; // Indicates a new format for a specific record type; usually 0
   public int reserve; // Reserved for future use
   public short recordLength; // Record Length following header (in bytes)

14     /**
   * Constructor
   */
15     public CHSRecord(FileInputStream fis){

16         fill(fis);
17     }
   /**
   * fills all fields of the Common Header Section Record
   */
18     public void fill(FileInputStream fis) {

19         super.fillBuffer(fis,12);
         langCode = new Integer(buffer[0]).byteValue();
20         recordType = new Integer(buffer[1]+(buffer[2]*256)).shortValue(); //xchanges byte 2 and 3; shifts
byte3 left
         sysadateLevel = new Integer(buffer[3]).byteValue();
21         contFlag = new Integer(buffer[4]).byteValue();
         editionLevel = new Integer(buffer[5]).byteValue();

```

```

1      recordLength = new Integer(buffer[10]+(buffer[11]*256)).shortValue();
2      }
3      /**
4      * returns length of the following record
5      */
6      public short getrecordLength(){
7          return(recordLength);
8      }
9      /**
10     * returns type of the following record in decimal
11     */
12     public short getrecordType(){
13         return(recordType);
14     }
15     /**
16     * prints out attributes of CHSRecord
17     */
18     public void print(){
19         trace("langCode " + langCode + ";recordType " +recordType+ " ;sysadataLevel "+sysadataLevel+
20         ";contFlag "+contFlag+ ";editionLevel "+editionLevel+ ";recordLength "+recordLength);
21     }
22     }
23     /**
24     * Compilation Unit Start/End Record - x0002
25     * 8 bytes long
26     */
27     import java.io.*;
28
29     class CURecord extends Record{
30         short type;
31     }
32     /**
33     * CURecord constructor
34     */
35     public CURecord(FileInputStream fis) {
36         fill(fis);
37     }
38     /**
39     * fills all needed fields of the Compilation Unit Start/End Record
40     */
41     public void fill(FileInputStream fis) {
42         super.fillBuffer(fis,8);
43
44         type = new Integer(buffer[0]+(buffer[1]*256)).shortValue();
45     }
46     /**
47     * returns the type of the CURecord
48     * compilation unit types are: x0000=Start, x0001=end
49     */
50     public int getType() {
51         return type;
52     }
53     }

```

```

1  /**
2   * SymbolRecord: Common header Section - x0001
3   * variable lenght, contains description of all symbols
4   *
5   * All get methods are used to fill the DataItems in the XMI Document.
6   */
7
8  import java.math.*;
9  class SymbolRecord extends Record {
10     //fields of the Common Header Section
11     int symbolId;
12     byte level;
13     byte symbolType;
14     byte symbolAttribute;
15     byte[] clauses = new byte[1];
16     BigInteger clausesB;
17     byte[] flags1 = new byte[1];
18     BigInteger flags1B;
19     int size;
20     int parentId;
21     int redefinedId;
22     short symbolNameLen;
23     String symbolName;
24
25     //symbolAttributeValues are put into the attribute tag of the DataItem
26     static String symbolAttributeValues[] = {
27         "",
28         "Numeric",
29         "Alphanumeric",
30         "Group",
31         "Pointer",
32         "IndexDataItem",
33         "IndexName",
34         "Condition",
35         "", "", "", "", "", "",
36         "File",
37         "SortFile",
38         "", "", "", "", "",
39         "ClassName",
40         "ObjectReference"};
41
42     /**
43     * Constructor
44     */
45     public SymbolRecord(java.io.FileInputStream fis, int length) {
46         fill(fis, length);
47     }
48     /**
49     * fills needed fields of Symbol Record
50     */
51     public void fill(java.io.FileInputStream fis, int length) {
52         super.fillBuffer(fis,length);

```

```

1      symbolId = new Integer(buffer[0] + buffer[1]*16*16 + buffer[2]*16*16*16*16 +
buffer[3]*16*16*16*16*16*16).intValue();
2      level = new Integer(buffer[8]).byteValue();
3      symbolType = new Integer(buffer[10]).byteValue();
      symbolAttribute = new Integer(buffer[11]).byteValue();
4      clauses[0] = new Integer(buffer[12]).byteValue();
      clausesB = new BigInteger(clauses);
      flags1[0] = new Integer(buffer[13]).byteValue();
5      flags1B = new BigInteger(flags1);
      size = new Integer(buffer[20] + buffer[21]*16*16 + buffer[22]*16*16*16*16 +
6      buffer[23]*16*16*16*16*16*16).intValue();
      parentId = new Integer(buffer[44] + buffer[45]*16*16 + buffer[46]*16*16*16*16 +
7      buffer[47]*16*16*16*16*16*16).intValue();
      redefinedId = new Integer(buffer[48] + buffer[49]*16*16 + buffer[50]*16*16*16*16 +
8      buffer[51]*16*16*16*16*16*16).intValue();
      symbolNameLen = new Integer(buffer[90] + buffer[91]*16*16).shortValue();
9
      char[] temp = new char[symbolNameLen];
      int j=0;
      for(int i=104;i<(104+symbolNameLen);i++){
10          temp[j]=(char)buffer[i];
          j++;
      }
      symbolName = new String(temp);
11
      }
      public String getAttribute(){
          return symbolAttributeValues[symbolAttribute];
12      }
      public String getLength(){
          return new Integer(size).toString() ;
13      }
      public int getLevel(){
          return level;
14      }
      }
      public String getName(){
          return symbolName;
15      }
      }
      public int getParentId(){
          return parentId;
16      }
      }
      public int getSize(){
          return size;
17      }
      }
      public int getSymbolId(){
          return symbolId;
18      }
      }
      public int getType(){
          return symbolType;
19      }
      }
20      /**
      * bit 7 is 1 if the symbol is redefined
21      * symbol is group if symbolAttribute == 3
      */

```

```
1 public boolean hasValue() {
      if(flags1B.testBit(7) || symbolAttribute == 3)
2         return false;
      else
3         return true;
4 }
5 public void print() {
      trace("symbolID:" + symbolId + " , level:" + level + " , parentId:" + parentId + " , size:" + size + " ,
6      redefined:" + flags1B.testBit(7) + " , symbolAttribute:" + symbolAttribute + " , redefinedId:" + redefinedId + " ,
      name:" + symbolName + " , hasValue:" + hasValue());
7 }
8 /**
9  * bit 5 is high if this symbol redefines another one
10 */
11
12 public boolean redefines() {
13     return clausesB.testBit(5);
14 }
15
16
17
18
19
20
21
```

Appendix B

Classes

In various embodiments, each record type is implemented as a class that understands how to handle the binary data from the SysAdata file 78 and provide the data to other classes via defined access methods. Figure 9 illustrates the relationship among the classes. A more detailed description on attributes and methods of each class follows. The source code of all classes may be found in Appendix A.

Class Super

This class offers methods and attributes that are needed by all classes.

Attributes

No Attributes defined.

Methods

- public static void trace(String s)

trace is used to trace error statements. It replaces Java's System.out.print().

Class Record

This class offers methods and attributes that are needed by all Record classes.

Attributes

- int[] buffer

Array of ints that stores the actual data from the SysAdata file 78.

- int length

Stores the size of the buffer array.

Methods

- public void fillBuffer(FileInputStream fis, int _length)

Fills the buffer array.

Class SysAdata

SysAdata wraps methods and attributes to read the SysAdata file 78 and create the XML document template 58. Initially, the complete SysAdata file 78 is processed and all data-entry symbols are saved in the symbolVector. Thereafter, the symbolVector is processed and an XMI model that resembles the document is created.

Attributes

- FileInputStream fis

Wraps the input adata file.

- CHSRecord chs_rec

Is used to temporarily save a CHSRecord.

- CUREcord cu_rec

Is used to temporarily save a CUREcord.

- Vector symbolVector

The symbolVector stores all SymbolRecords that are extracted from the adata file.

- Id model

model is the root element for the XMI toolkit object hierarchy.

- `int rootLength`

Stores the added up length of all `DataItem.VALUES`.

Methods

- `public static void main(String args[])`

The main method contains the actual loop, in which the SysAdata file is processed. The assembly of the model for the XML document template 58 is initiated here as well.

- `public static int addID(Id parent, int symbolNumber)`

Recursive method to assemble the model that is then saved as XML.

Parameters are:

- the parents Id, such that the next symbol can be added on the right level.
- the position number of the symbol to be added in the symbolVector.

`addID` returns the index of the next `SymbolRecord` object to be processed.

- `public static void skip(int length)`

Replaces `FileInputStream.skip()`.

Class CHSRecord

Throughout the processing of the SysAdata file 78, each common header section instantiates this class.

Attributes

A description of these attributes is found in Table 1.

- byte langCode
- short recordType
- byte sysadataLevel
- byte contFlag
- byte editionLevel
- public int reserve
- public short recordLength

Methods

- public CHSRecord(FileInputStream fis)
The constructor instantiates the CHSRecord class. The FileInputStream object is handed over from the calling instance.
- public void fill(FileInputStream fis)
Calls Record.fillBuffer() and then extracts the binary data stored in buffer[] into the referring attribute.
- public short getrecordLength()
Returns the length of the following record. Is needed in one embodiment to process the input file correctly.

- public short getrecordType()

Returns the record type.

Class CUREcord

A compilation unit start/end record instantiates this class. The CUREcord class is used to control the processing of the SysAdata file 78. In certain embodiments, if getType() returns 1, the processing is stopped.

Attributes

- short type

Methods

- public CUREcord(FileInputStream fis)

The constructor instantiates the CUREcord class. The FileInputStream object is handed over from the calling instance.

- public void fill(FileInputStream fis)

Calls Record.fillBuffer() and then extracts the binary data stored in buffer[] into the refering attribute.

- public int getType()

Class SymbolRecord

Any symbol record found during the processing of the SysAdata file 78 instantiates this class.

Attributes

A detailed description of these attribute is given in Table 3. The clauses and the flags1 attribute are converted into BigIntegers and are therefore stored in an array with size one to provide the correct input to the BigInteger constructor. The BigInteger object is used, because it allows bit-operations on its value.

- int symbolId
- byte level
- byte symbolType
- byte symbolAttribute
- byte[] clauses = new byte[1]
- BigInteger clausesB
- byte[] flags1 = new byte[1]
- BigInteger flags1B
- int size
- int parentId
- int redefinedId
- short symbolNameLen

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

- String symbolName
- static String symbolAttributeValues[]

This array contains descriptive names for the DataItem.TYPE tag.

Appendix C

Document Type Definition (DTD)

```

<?xml version="1.0" encoding="UTF-8" ?>

<!-- XMI Automatic DTD Generation -->
<!-- Metamodel: DataItem -->

<!-- _____ -->
<!-- _____ -->
<!-- XMI is the top-level XML element for XMI transfer text -->
<!-- _____ -->

<ELEMENT XMI (XMI.header, XMI.content?, XMI.difference*,
             XMI.extensions*) >
<ATTLIST XMI
    xmi.version CDATA #FIXED "1.0"
    timestamp CDATA #IMPLIED
    verified (true | false) #IMPLIED
>

<!-- _____ -->
<!-- _____ -->
<!-- XMI.header contains documentation and identifies the model,
<!-- metamodel, and metamodel
<!-- _____ -->

<ELEMENT XMI.header (XMI.documentation?, XMI.model*, XMI.metamodel*,
                    XMI.metamodel*) >

<!-- _____ -->
<!-- _____ -->
<!-- documentation for transfer data -->
<!-- _____ -->

<ELEMENT XMI.documentation (#PCDATA | XML.owner | XML.contact |
                           XML.longDescription | XML.shortDescription |
                           XML.exporter | XML.exporterVersion |
                           XML.notice)* >

<ELEMENT XMI.owner ANY >

<ELEMENT XMI.contact ANY >

<ELEMENT XMI.longDescription ANY >

<ELEMENT XMI.shortDescription ANY >

```

```

1  <!ELEMENT XML.exporter ANY >
2  <!ELEMENT XML.exporterVersion ANY >
3  <!ELEMENT XML.exporterID ANY >
4  <!ELEMENT XML.notice ANY >
5  <!-- _____ -->
6  <!-- _____ -->
7  <!-- XMI.element.att defines the attributes that each XML element
8  <!-- that corresponds to a metamodel class must have to conform to
9  <!-- the XMI specification.
10 <!-- _____ -->
11 <!-- _____ -->
12 <!-- _____ -->
13 <!-- XMI.link.att defines the attributes that each XML element that
14 <!-- corresponds to a metamodel class must have to enable it to
15 <!-- function as a simple XLink as well as refer to model
16 <!-- constructs within the same XMI file.
17 <!-- _____ -->
18 <!-- _____ -->
19 <!-- XMI.link.att
20 <!-- 'xmi:link CDATA #IMPLIED inline (true | false) #IMPLIED
21 <!-- actuate (show | user) #IMPLIED href CDATA #IMPLIED role
22 <!-- CDATA #IMPLIED title CDATA #IMPLIED show (embed | replace
23 <!-- | new) #IMPLIED behavior CDATA #IMPLIED xmi:idref IDREF
24 <!-- #IMPLIED xmi:uuidref CDATA #IMPLIED' >
25 <!-- _____ -->
26 <!-- _____ -->
27 <!-- XMI.model identifies the model(s) being transferred
28 <!-- _____ -->
29 <!-- _____ -->
30 <!-- XMI.model ANY >
31 <!-- ATTLIST XMI.model
32 <!-- %XMI.link.att;
33 <!-- xmi.name CDATA #REQUIRED
34 <!-- xmi.version CDATA #IMPLIED
35 <!-- >
36 <!-- _____ -->
37 <!-- _____ -->
38 <!-- XMI.metamodel identifies the metamodel(s) for the transferred
39 <!-- data
40 <!-- _____ -->

```

```

1  <!-- _____ -->
2  <!ELEMENT XML.metamodel ANY >
3  <!ATTLIST XML.metamodel
4      %XML.link.att;
5      xmi.name CDATA #REQUIRED
6      xmi.version CDATA #IMPLIED
7  >
8  <!-- _____ -->
9  <!-- _____ -->
10 <!-- XML.metamodel identifies the metametamodel(s) for the
11 <!-- transferred data
12 <!-- _____ -->
13 <!ELEMENT XML.metametamodel ANY >
14 <!ATTLIST XML.metametamodel
15     %XML.link.att;
16     xmi.name CDATA #REQUIRED
17     xmi.version CDATA #IMPLIED
18 >
19 <!-- _____ -->
20 <!-- _____ -->
21 <!-- XML.content is the actual data being transferred
22 <!-- _____ -->
23 <!ELEMENT XML.content ANY >
24 <!-- _____ -->
25 <!-- _____ -->
26 <!-- XML.extensions contains data to transfer that does not conform
27 <!-- to the metamodel(s) in the header
28 <!-- _____ -->
29 <!ELEMENT XML.extensions ANY >
30 <!ATTLIST XML.extensions
31     xmi.extender CDATA #REQUIRED
32 >
33 <!-- _____ -->
34 <!-- _____ -->
35 <!-- extension contains information related to a specific model
36 <!-- construct that is not defined in the metamodel(s) in the header
37 <!-- _____ -->
38 <!ELEMENT XML.extension ANY >
39 <!ATTLIST XML.extension
40     %XML.element.att;
41     %XML.link.att;

```



```

1      xmi.extender CDATA #REQUIRED
      xmi.extenderID CDATA #REQUIRED
2  >
3  <!-- _____ -->
4  <!-- _____ -->
      <!-- XMI.difference holds XML elements representing differences to a -->
      <!-- base model -->
5  <!-- _____ -->
6  <!ELEMENT XMI.difference (XMI.difference | XMI.delete | XMI.add |
      XMI.replace)* >
7  <!-- _____ -->
      <!-- XMI.delete represents a deletion from a base model -->
8  <!-- _____ -->
9  <!-- _____ -->
10 <!ELEMENT XMI.delete EMPTY >
11 <!-- _____ -->
      <!-- XMI.add represents an addition to a base model -->
12 <!-- _____ -->
13 <!-- _____ -->
14 <!-- _____ -->
15 <!-- _____ -->
      <!-- XMI.replace represents the replacement of a model construct -->
16 <!-- _____ -->
      <!-- with another model construct in a base model -->
17 <!-- _____ -->
18 <!-- _____ -->
19 <!-- _____ -->
20 <!-- _____ -->
21 <!-- _____ -->

```

```

1      xmi.position CDATA "-1"
2      >
3      <!-- _____ -->
4      <!-- _____ -->
5      <!-- XMI.reference may be used to refer to data types not defined in
6      <!-- the metamodel
7      <!-- _____ -->
8      <!-- _____ -->
9      <!-- _____ -->
10     <!-- _____ -->
11     <!-- This section contains the declaration of XML elements
12     <!-- representing data types
13     <!-- _____ -->
14     <!-- _____ -->
15     <!-- _____ -->
16     <!-- _____ -->
17     <!-- _____ -->
18     <!-- _____ -->
19     <!-- _____ -->
20     <!-- _____ -->
21     <!-- _____ -->

```

```

1          XMI.CorbaTcChar | XMI.CorbaTcWchar |
2          XMI.CorbaTcOctet | XMI.CorbaTcAny |
3          XMI.CorbaTcTypeCode | XMI.CorbaTcPrincipal |
4          XMI.CorbaTcNull | XMI.CorbaTcVoid |
5          XMI.CorbaTcLongLong |
6          XMI.CorbaTcLongDouble) >
<!ATTLIST XMI.CorbaTypeCode
7          %XMI.element.att;
8          >
9
10         <!ELEMENT XMI.CorbaTcAlias (XMI.CorbaTypeCode) >
11         <!ATTLIST XMI.CorbaTcAlias
12             xmi.tcName CDATA #REQUIRED
13             xmi.tcId CDATA #IMPLIED
14         >
15
16         <!ELEMENT XMI.CorbaTcStruct (XMI.CorbaTcField)* >
17         <!ATTLIST XMI.CorbaTcStruct
18             xmi.tcName CDATA #REQUIRED
19             xmi.tcId CDATA #IMPLIED
20         >
21
22         <!ELEMENT XMI.CorbaTcField (XMI.CorbaTypeCode) >
23         <!ATTLIST XMI.CorbaTcField
24             xmi.tcName CDATA #REQUIRED
25         >
26
27         <!ELEMENT XMI.CorbaTcSequence (XMI.CorbaTypeCode |
28             XMI.CorbaRecursiveType) >
29         <!ATTLIST XMI.CorbaTcSequence
30             xmi.tcLength CDATA #REQUIRED
31         >
32
33         <!ELEMENT XMI.CorbaRecursiveType EMPTY >
34         <!ATTLIST XMI.CorbaRecursiveType
35             xmi.offset CDATA #REQUIRED
36         >
37
38         <!ELEMENT XMI.CorbaTcArray (XMI.CorbaTypeCode) >
39         <!ATTLIST XMI.CorbaTcArray
40             xmi.tcLength CDATA #REQUIRED
41         >
42
43         <!ELEMENT XMI.CorbaTcObjRef EMPTY >
44         <!ATTLIST XMI.CorbaTcObjRef
45             xmi.tcName CDATA #REQUIRED
46             xmi.tcId CDATA #IMPLIED
47         >
48
49         <!ELEMENT XMI.CorbaTcEnum (XMI.CorbaTcEnumLabel) >

```

```
1  <!ATTLIST XMI.CorbaTcEnum
   xmi.tcName CDATA #REQUIRED
   xmi.tcId   CDATA #IMPLIED
2  >
3  <!ELEMENT XMI.CorbaTcEnumLabel EMPTY >
   <!ATTLIST XMI.CorbaTcEnumLabel
4     xmi.tcName CDATA #REQUIRED
   >
5  <!ELEMENT XMI.CorbaTcUnionMbr (XMI.CorbaTypeCode, XMI.any) >
   <!ATTLIST XMI.CorbaTcUnionMbr
6     xmi.tcName CDATA #REQUIRED
   >
7  <!ELEMENT XMI.CorbaTcUnion (XMI.CorbaTypeCode, XMI.CorbaTcUnionMbr*) >
   <!ATTLIST XMI.CorbaTcUnion
8     xmi.tcName CDATA #REQUIRED
     xmi.tcId   CDATA #IMPLIED
9  >
10 <!ELEMENT XMI.CorbaTcExcept (XMI.CorbaTcField)* >
   <!ATTLIST XMI.CorbaTcExcept
11     xmi.tcName CDATA #REQUIRED
     xmi.tcId   CDATA #IMPLIED
   >
12 <!ELEMENT XMI.CorbaTcString EMPTY >
   <!ATTLIST XMI.CorbaTcString
13     xmi.tcLength CDATA #REQUIRED
   >
14 <!ELEMENT XMI.CorbaTcWstring EMPTY >
   <!ATTLIST XMI.CorbaTcWstring
15     xmi.tcLength CDATA #REQUIRED
   >
16 <!ELEMENT XMI.CorbaTcFixed EMPTY >
   <!ATTLIST XMI.CorbaTcFixed
17     xmi.tcDigits CDATA #REQUIRED
     xmi.tcScale  CDATA #REQUIRED
18 >
19 <!ELEMENT XMI.CorbaTcShort EMPTY >
20 <!ELEMENT XMI.CorbaTcLong EMPTY >
   <!ELEMENT XMI.CorbaTcUshort EMPTY >
21 <!ELEMENT XMI.CorbaTcUlong EMPTY >
```

```

1  <!ELEMENT XMI.CorbaTcFloat EMPTY >
2  <!ELEMENT XMI.CorbaTcDouble EMPTY >
3  <!ELEMENT XMI.CorbaTcBoolean EMPTY >
4  <!ELEMENT XMI.CorbaTcChar EMPTY >
5  <!ELEMENT XMI.CorbaTcWchar EMPTY >
6  <!ELEMENT XMI.CorbaTcOctet EMPTY >
7  <!ELEMENT XMI.CorbaTcAny EMPTY >
8  <!ELEMENT XMI.CorbaTcTypeCode EMPTY >
9  <!ELEMENT XMI.CorbaTcPrincipal EMPTY >
10 <!ELEMENT XMI.CorbaTcNull EMPTY >
11 <!ELEMENT XMI.CorbaTcVoid EMPTY >
12 <!ELEMENT XMI.CorbaTcLongLong EMPTY >
13 <!ELEMENT XMI.CorbaTcLongDouble EMPTY >
14 <!-- _____ -->
15 <!-- _____ -->
16 <!-- METAMODEL: DataItem _____ -->
17 <!-- _____ -->
18 <!-- _____ -->
19 <!-- METAMODEL CLASS: DataItem _____ -->
20 <!-- _____ -->
21 <!-- _____ -->
22 <!-- _____ -->
23 <!-- _____ -->
24 <!-- _____ -->
25 <!-- _____ -->
26 <!-- _____ -->
27 <!-- _____ -->
28 <!-- _____ -->
29 <!-- _____ -->
30 <!-- _____ -->
31 <!-- _____ -->
32 <!-- _____ -->
33 <!-- _____ -->
34 <!-- _____ -->
35 <!-- _____ -->
36 <!-- _____ -->
37 <!-- _____ -->
38 <!-- _____ -->
39 <!-- _____ -->
40 <!-- _____ -->
41 <!-- _____ -->
42 <!-- _____ -->
43 <!-- _____ -->
44 <!-- _____ -->
45 <!-- _____ -->
46 <!-- _____ -->
47 <!-- _____ -->
48 <!-- _____ -->
49 <!-- _____ -->
50 <!-- _____ -->
51 <!-- _____ -->
52 <!-- _____ -->
53 <!-- _____ -->
54 <!-- _____ -->
55 <!-- _____ -->
56 <!-- _____ -->
57 <!-- _____ -->
58 <!-- _____ -->
59 <!-- _____ -->
60 <!-- _____ -->
61 <!-- _____ -->
62 <!-- _____ -->
63 <!-- _____ -->
64 <!-- _____ -->
65 <!-- _____ -->
66 <!-- _____ -->
67 <!-- _____ -->
68 <!-- _____ -->
69 <!-- _____ -->
70 <!-- _____ -->
71 <!-- _____ -->
72 <!-- _____ -->
73 <!-- _____ -->
74 <!-- _____ -->
75 <!-- _____ -->
76 <!-- _____ -->
77 <!-- _____ -->
78 <!-- _____ -->
79 <!-- _____ -->
80 <!-- _____ -->
81 <!-- _____ -->
82 <!-- _____ -->
83 <!-- _____ -->
84 <!-- _____ -->
85 <!-- _____ -->
86 <!-- _____ -->
87 <!-- _____ -->
88 <!-- _____ -->
89 <!-- _____ -->
90 <!-- _____ -->
91 <!-- _____ -->
92 <!-- _____ -->
93 <!-- _____ -->
94 <!-- _____ -->
95 <!-- _____ -->
96 <!-- _____ -->
97 <!-- _____ -->
98 <!-- _____ -->
99 <!-- _____ -->
100 <!-- _____ -->

```

1 <!ATTLIST DataItem.hasValue
xmi.value (true | false) #REQUIRED
2 >
3 <!ELEMENT DataItem.Value (#PCDATA | XML.reference)* >
4 <!ELEMENT DataItem.parent (DataItem)? >
5 <!ELEMENT DataItem (DataItem.Name?, DataItem.Type?, DataItem.Length?,
DataItem.hasValue?, DataItem.Value?, XML.extension*,
6 DataItem.parent?, DataItem.child*)? >
7 <!ATTLIST DataItem
%XML.element.att;
%XML.link.att;
8 >
9
10
11
12
13
14
15
16
17
18
19
20
21

Claims

1.
A computer-implemented method for representing IMS messages as XML documents, the method comprising:

generating an XML document template from an IMS message definition; and
merging an IMS message with the generated template to produce a
corresponding XML document.

2. The method of claim 1, wherein the generating step comprises:
obtaining an IMS message definition;
obtaining a DTD for representing arbitrary IMS message definitions;
compiling the IMS message definition with an option configured to produce
an associated data (Adata) file; and
parsing the Adata file using the DTD to generate an XML document
template corresponding to the IMS message definition.

3. The method of claim 2, wherein the generating step comprises:
obtaining an IMS message definition;
obtaining a DTD for representing arbitrary IMS message definitions; and
parsing the IMS message definition using the DTD to generate an XML
document template corresponding to the IMS message definition.

1 4. The method of claim 2, wherein the Adata file comprises at least one
2 IMS message definition in a relatively language independent format compared with
3 program source code.

4
5 5. The method of claim 2, wherein obtaining the IMS message definition
6 comprises:

7 extracting the IMS message definition from one of an application source code
8 file and a copy file.

9
10 6. The method of claim 2, wherein the step of obtaining the DTD
11 comprises:

12 creating a UML object model for representing arbitrary IMS message
13 definitions; and
14 processing the object model using an XMI utility to generate the DTD.

15
16 7. The method of claim 2, wherein the merging step comprises:
17 identifying a placeholder within the XML document template for receiving
18 a corresponding value from the IMS message;
19 reading the value from the IMS message; and
20 inserting the value into a location within the XML document template
21 indicated by the placeholder.

1 8. The method of claim 7, wherein the placeholder comprises an XML
2 tag.

3
4 9. The method of claim 7, wherein the identifying step comprises:
5 checking the placeholder for an associated tag indicating that a
6 corresponding value exists within the IMS message.

7
8 10. The method of claim 7, wherein at least one placeholder has an
9 associated tag indicating the size of the corresponding value within the IMS
10 message, the reading step comprising:

11 reading a portion of the IMS message corresponding to the indicated size.

12
13 ~~11.~~ A system for representing IMS messages as XML documents, the
14 system comprising:

15 a template generation module configured to generate an XML document

16 template from an IMS message definition; and

17 a merging module configured to merge an IMS message with the generated
18 template to produce a corresponding XML document.

19
20 12. The system of claim 11, wherein the template generating module
21 comprises:

1 a compiler configured to compile an IMS message definition with an option
2 configured to produce an associated data (Adata) file; and
3 a parser configured to parse the Adata file using a DTD for representing
4 arbitrary IMS message definitions to generate an XML document
5 template corresponding to the IMS message definition.
6

7 13. The system of claim 12, wherein the template generating module
8 comprises:

9 a parser configured to obtain a DTD for representing arbitrary IMS message
10 definitions and parse the IMS message definition using the DTD to
11 generate an XML document template corresponding to the IMS
12 message definition.
13

14 14. The system of claim 12, wherein the Adata file comprises at least one
15 IMS message definition in a relatively language independent format compared with
16 program source code.
17

18 15. The system of claim 12, further comprising:
19 a message definition extractor configured to extract the IMS message
20 definition from one of an application source code file and a copy file.
21

1 16. The system of claim 12, further comprising:
2 a modeling tool configured to create a UML object model for representing
3 arbitrary IMS message definitions; and
4 an XMI utility for generating the DTD from the UML object model.
5

6 17. The system of claim 12, wherein the merging module is further
7 configured to identify a placeholder within XML document template for receiving
8 a corresponding value from the IMS message; read the value from the IMS message;
9 and insert the value into a location within the XML document template indicated
10 by the placeholder.
11

12 18. The system of claim 17, wherein the placeholder comprises an XML
13 tag.
14

15 19. The system of claim 17, wherein at least one placeholder comprises an
16 associated tag indicating whether a corresponding value exists within the IMS
17 message.
18

19 20. The system of claim 7, wherein at least one placeholder has an
20 associated tag indicating the size of the corresponding value within the IMS
21 message.

1 21. An article of manufacture comprising a program storage medium
2 readable by a processor and embodying one or more instructions executable by the
3 processor to perform a computer-implemented method for representing IMS
4 messages as XML documents, the method comprising:

5 generating an XML document template from an IMS message definition; and
6 merging an IMS message with the generated template to produce a
7 corresponding XML document.

8
9 22. The article of claim 21, wherein the generating step comprises:
10 obtaining an IMS message definition;
11 obtaining a DTD for representing arbitrary IMS message definitions;
12 compiling the IMS message definition with an option configured to produce
13 an associated data (Adata) file; and
14 parsing the Adata file using the DTD to generate an XML document
15 template corresponding to the IMS message definition.

16
17 23. The article of claim 22, wherein the IMS message definition comprises
18 program source code in a language selected from the group consisting of COBOL,
19 PL/I, Assembler, and Pascal.

1 24. The article of claim 22, wherein the Adata file comprises at least one
2 IMS message definition in a relatively language independent format compared with
3 program source code.

4
5 25. The article of claim 22, wherein obtaining the IMS message definition
6 comprises:

7 extracting the IMS message definition from one of an application source code
8 file and a copy file.

9
10 26. The article of claim 22, wherein the step of obtaining the DTD
11 comprises:

12 creating a UML object model for representing arbitrary IMS message
13 definitions; and
14 processing the object model using an XMI utility to generate the DTD.

15
16 27. The article of claim 22, wherein the merging step comprises:
17 identifying a placeholder within XML document template for receiving a
18 corresponding value from the IMS message;
19 reading the value from the IMS message; and
20 inserting the value into a location within the XML document template
21 indicated by the placeholder.

1 28. The article of claim 27, wherein the placeholder comprises an XML tag.

2

3 29. The article of claim 27, wherein the identifying step comprises:
4 checking the placeholder for an associated tag indicating that a
5 corresponding value exists within the IMS message.

6

7 30. The article of claim 27, wherein at least one placeholder has an
8 associated tag indicating the size of the corresponding value within the IMS
9 message, the reading step comprising:
10 reading a portion of the IMS message corresponding to the indicated size.

11

12

13

14

15

16

17

18

19

20

21

1 REPRESENTING IMS MESSAGES AS XML DOCUMENTS

2 ABSTRACT OF THE DISCLOSURE

3 A computer-implemented method for representing IMS messages as XML
4 documents includes generating an XML document template from an IMS message
5 definition and merging an IMS message with the generated template to produce a
6 corresponding XML document. A system for representing IMS messages as XML
7 documents includes a template generation module configured to generate an XML
8 document template from an IMS message definition and a merging module
9 configured to merge an IMS message with the generated template to produce a
10 corresponding XML document.
11
12
13
14
15
16
17
18
19
20
21

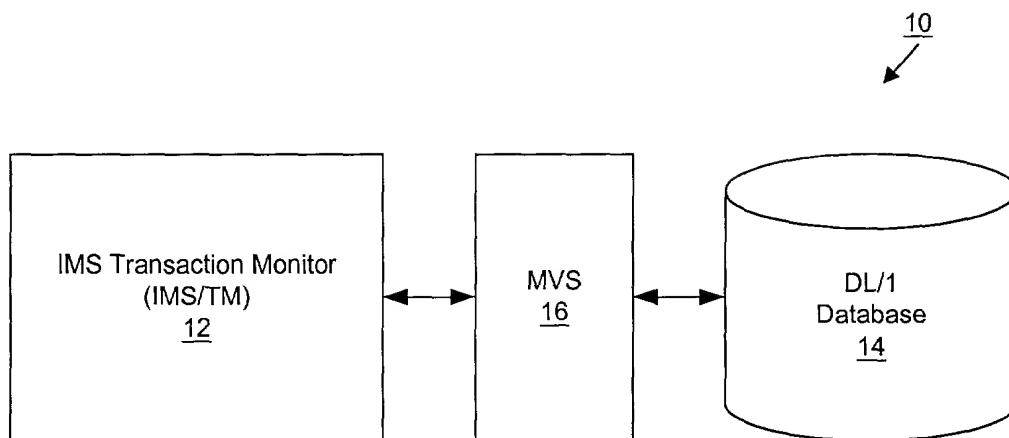


Fig. 1
(prior art)

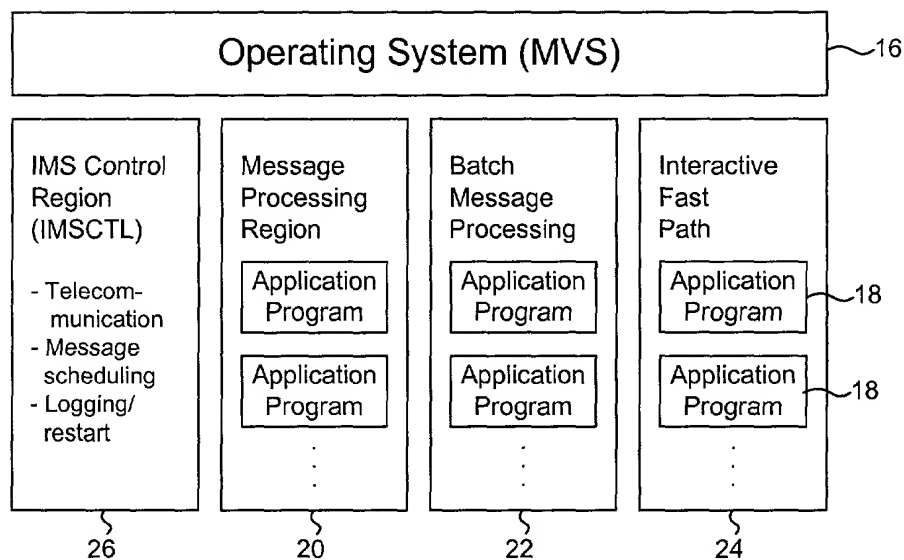


Fig. 2
(prior art)

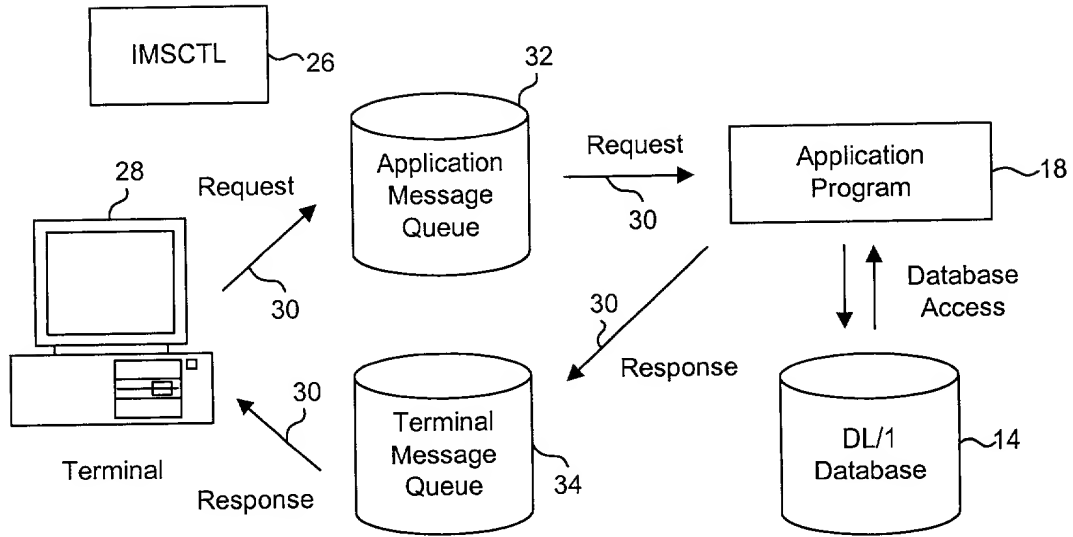


Fig. 3
(prior art)

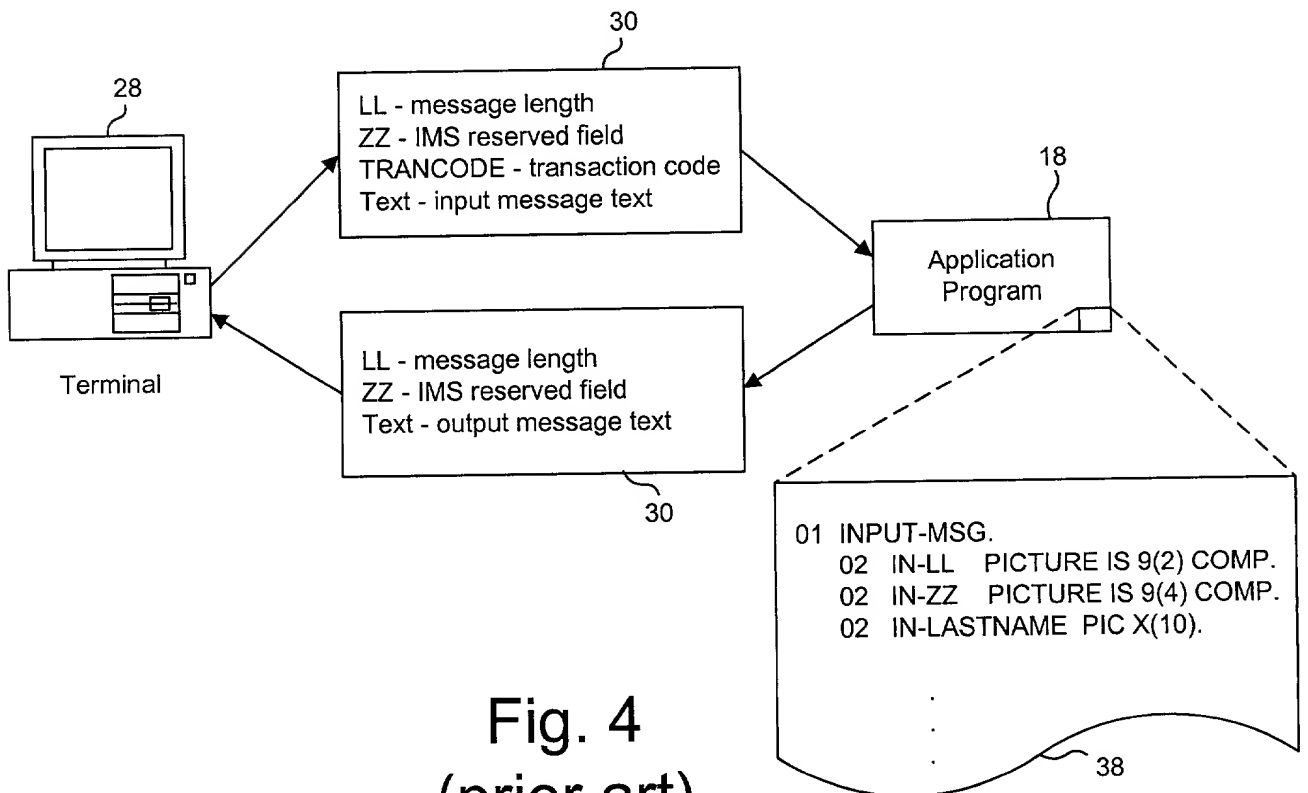


Fig. 4
(prior art)

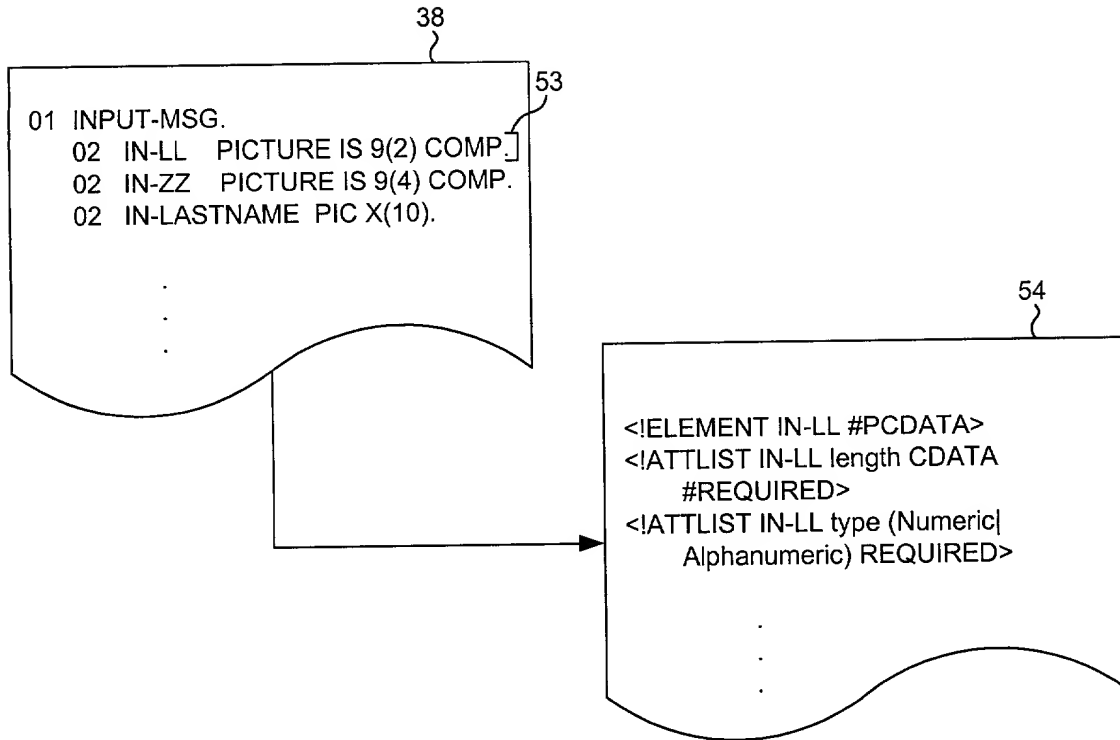


Fig. 5

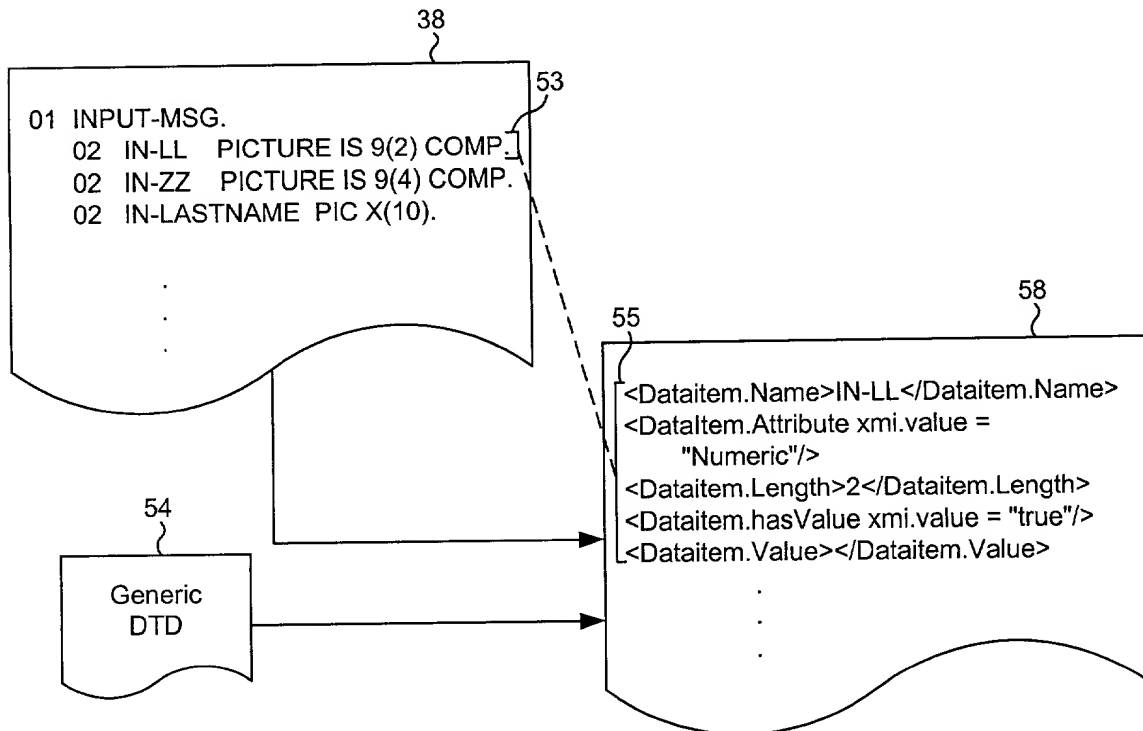


Fig. 6

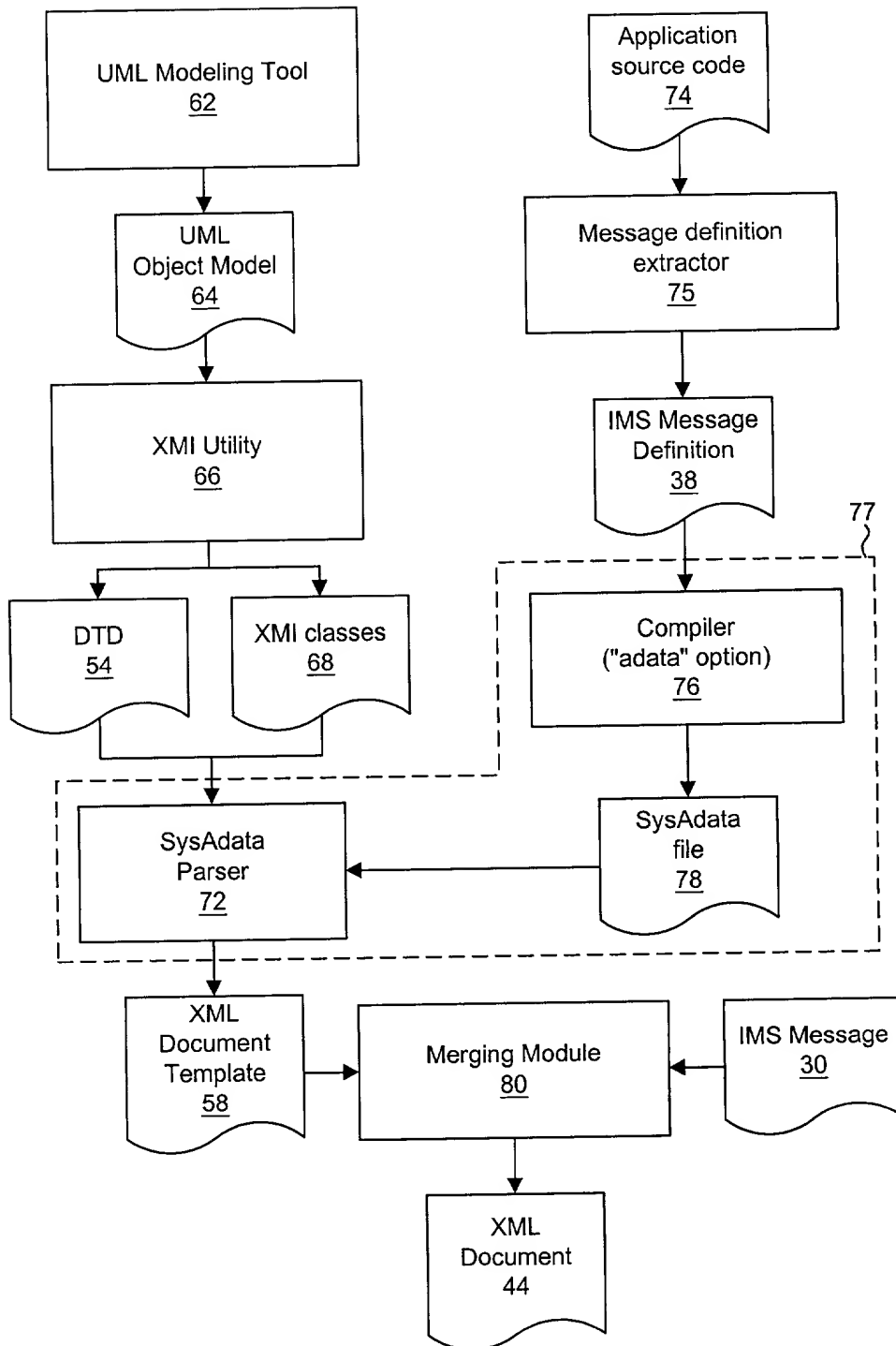


Fig. 7

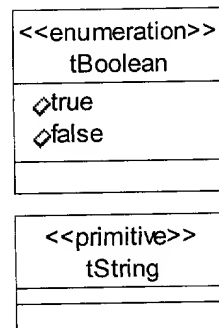
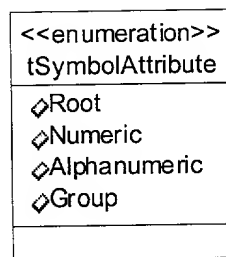
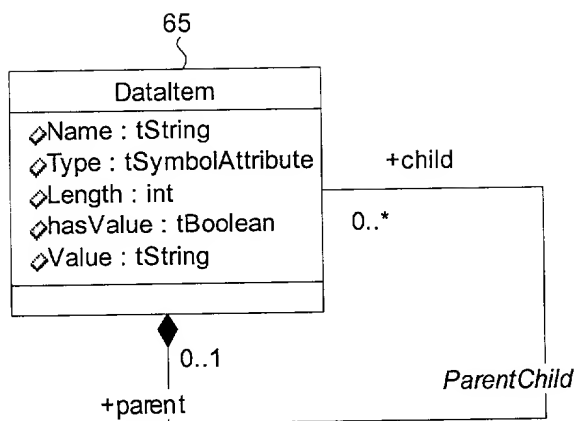


Fig. 8

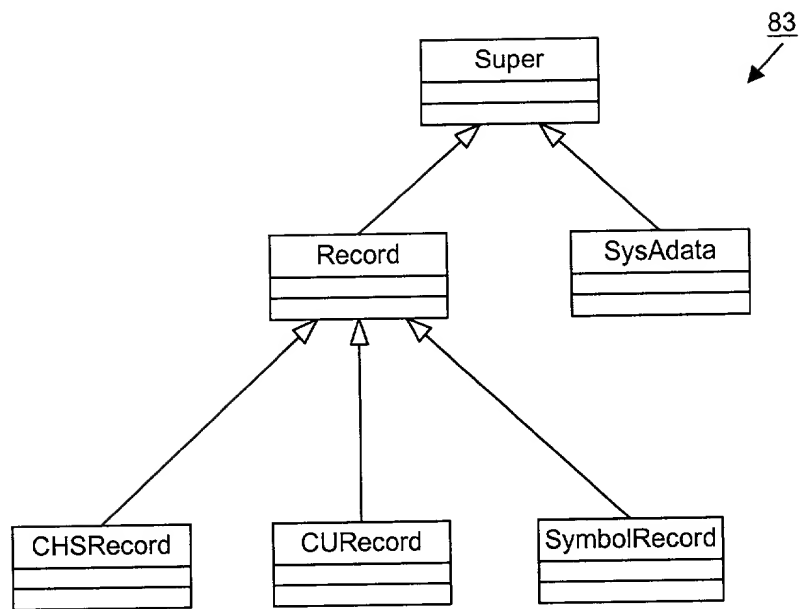


Fig. 9

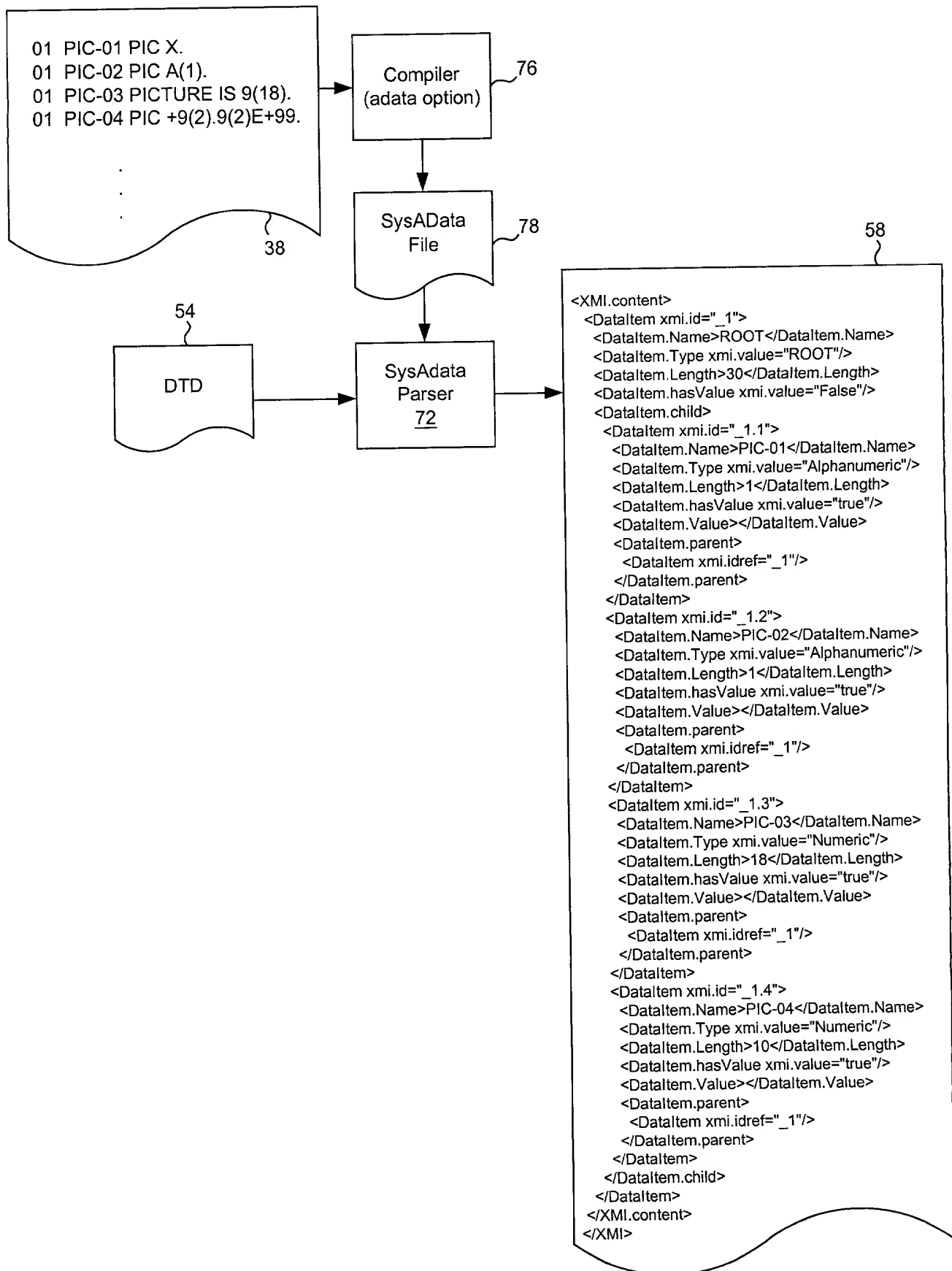


Fig. 10

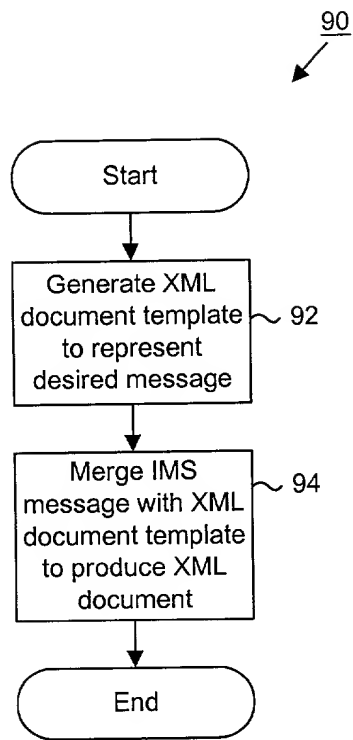


Fig. 11

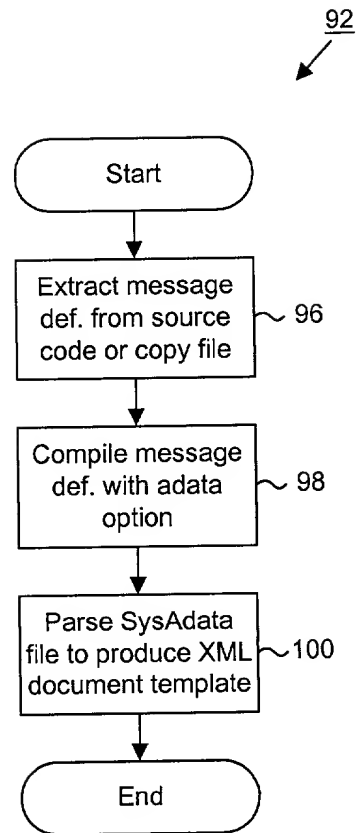


Fig. 12

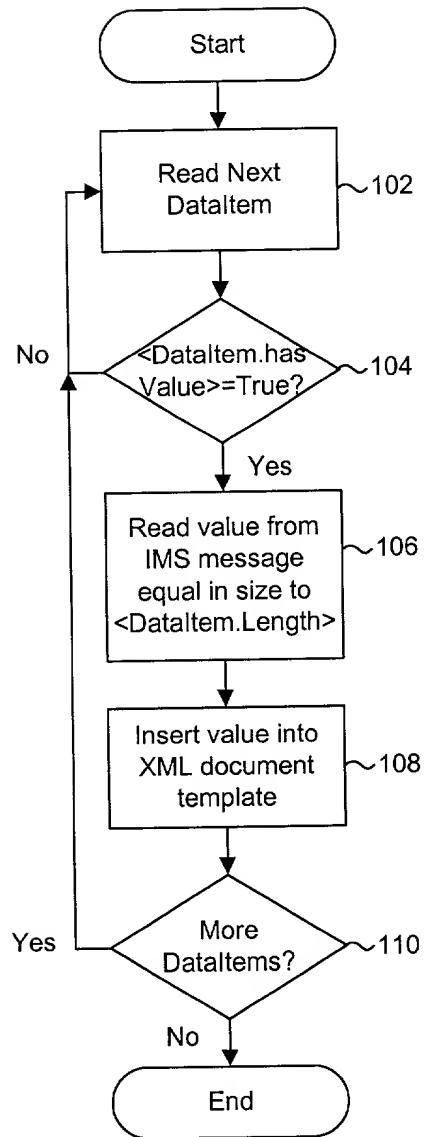


Fig. 13

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence and citizenship are as stated below next to my name;

I believe I am the original, first and joint inventor of the subject matter which is claimed and for which a patent is sought on the invention entitled

REPRESENTING IMS MESSAGES AS XML DOCUMENTS

the specification of which (check one)

 X is attached hereto.
 _____ was filed on _____
 _____ as Application Serial No. _____
 _____ and was amended on _____ (if applicable).

I hereby stat that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, united States Code, Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)

Priority Claimed

 none Yes No
 (Number) (Country) (Day/Month/Year filed)

I hereby claim the benefit under Title 35, Untied States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56, which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

 60/151,479 August 30, 1999 Pending
 (Application Serial No.) (Filing Date) (Status) (patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. (list name and registration number)

Romualdas Strimaitis: 35,697
 Timothy M. Farrell: 37,321
 Ingrid M. Foerster: 36,511
 Prentiss W. Johnson: 33,123
 Christopher A. Hughes: 26,914
 John E. Hoel: 26,279
 Edward A. Pennington: 32,588
 Joseph C. Redmond, Jr.: 18,753
 Craig J. Madson: 29,407
 L. Craig Metcalf: 31,398

Evan R. Witt: 32,512
 A. John Pate: 36,234
 Gary D.E. Pierce: 38,019
 David B. Fonda: 39,672
 John R. Thompson: 40,842
 Barton W. Giddings: 41,036
 Hal D. Baird: 42,284
 Kory D. Christensen: 43,548

Send correspondence to: Kory D. Christensen
 MADSON & METCALF
 15 West South Temple, Suite 900
 Salt Lake City, Utah 84101
 Telephone: (801) 537-1700

Full name of sole or first joint-inventor: Jan Burchhardt

Inventor's signature: J. Burchhardt Date: May 123/2000

Residence: Fraasstr 5, D-70184, Stuttgart, Federal Republic of Germany

Citizenship: German

Post Office Address: Same

Full name of second joint-inventor: Shyh-Mei F. Ho

Inventor's signature: _____ Date: _____

Residence: 10375 Moretti Drive, Cupertino, California 95014

Citizenship: United States of America

Post Office Address: Same

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence and citizenship are as stated below next to my name;

I believe I am the original, first and joint inventor of the subject matter which is claimed and for which a patent is sought on the invention entitled

REPRESENTING IMS MESSAGES AS XML DOCUMENTS

the specification of which (check one)

 X is attached hereto.
 was filed on _____
as Application Serial No. _____
and was amended on _____ (if applicable).

I hereby stat that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, united States Code, Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Claimed
<u> none </u>	<u> </u>	<u> </u>	<u> </u> Yes <u> </u> No
(Number)	(Country)	(Day/Month/Year filed)	

I hereby claim the benefit under Title 35, Untied States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56, which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

<u> 60/151,479 </u>	<u> August 30, 1999 </u>	<u> Pending </u>
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. (list name and registration number)

Romualdas Strimaitis: 35,697
 Timothy M. Farrell: 37,321
 Ingrid M. Foerster: 36,511
 Prentiss W. Johnson: 33,123
 Christopher A. Hughes: 26,914
 John E. Hoel: 26,279
 Edward A. Pennington: 32,588
 Joseph C. Redmond, Jr.: 18,753
 Craig J. Madson: 29,407
 L. Craig Metcalf: 31,398

Evan R. Witt: 32,512
 A. John Pate: 36,234
 Gary D.E. Pierce: 38,019
 David B. Fonda: 39,672
 John R. Thompson: 40,842
 Barton W. Giddings: 41,036
 Hal D. Baird: 42,284
 Kory D. Christensen: 43,548

Send correspondence to:

Kory D. Christensen
 MADSON & METCALF
 15 West South Temple, Suite 900
 Salt Lake City, Utah 84101
 Telephone: (801) 537-1700

Full name of sole or first joint-inventor: Jan Burchhardt

Inventor's signature:

Date:

Residence: Fraasstr 5, D-70184, Stuttgart, Federal Republic of Germany

Citizenship: German

Post Office Address: Same

Full name of second joint-inventor: Shyh-Mei F. Ho

Inventor's signature:

Shyh-Mei F. Ho

Date:

5/30/2000

Residence: 10375 Moretti Drive, Cupertino, California 95014

Citizenship: United States of America

Post Office Address: Same